

GT-VFPU: OPERATING SPEED TEST

SPECIAL TECHNICAL REPORT

REPORT NO. STR-0142-91-0012

April 16, 1991

**GUIDANCE, NAVIGATION AND CONTROL
DIGITAL EMULATION TECHNOLOGY LABORATORY**

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30322-0540

Contract Data Requirements List Item A004

Period Covered: Not Applicable

Type Report: As Required

**Reproduced From
Best Available Copy**

20010822 055

UL11695

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

RECEIVED
DEFENSE RESEARCH
AGENCY
7/10/91

DISCLAIMER

DISCLAIMER STATEMENT – The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

- (1) **DISTRIBUTION STATEMENT** – Approved for public release; distribution is unlimited.
- (2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013, October 1988.

GT-VFPU: OPERATING SPEED TEST

April 16, 1991

Author

Wei Siong Tan

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30322-0540

Eugene L. Sanders

USASDC

Contract Monitor

Cecil O. Alford

Georgia Tech

Project Director

Copyright 1990

Georgia Tech Research Corporation

Centennial Research Building

Atlanta, Georgia 30332

GT-VFPU Test Board

Wei Siong Tan

1. Introduction

The GT-VFPU test board was developed to characterize the operating speed of the GT-VFPU chip. This document presents the design of the test board, the testing strategy, and the test results.

2. Board Design

The architecture of the GT-VFPU test board is shown in Figure 1. The board was designed on a Multibus I board. A Multibus I to PC-AT interface board is used to connect the test board to a PC-AT host. The clock that drives the GT-VFPU chip is connected externally to a function generator. The power to the GT-VFPU chip is decoupled from the Multibus power plane. It is connected to an external power supply.

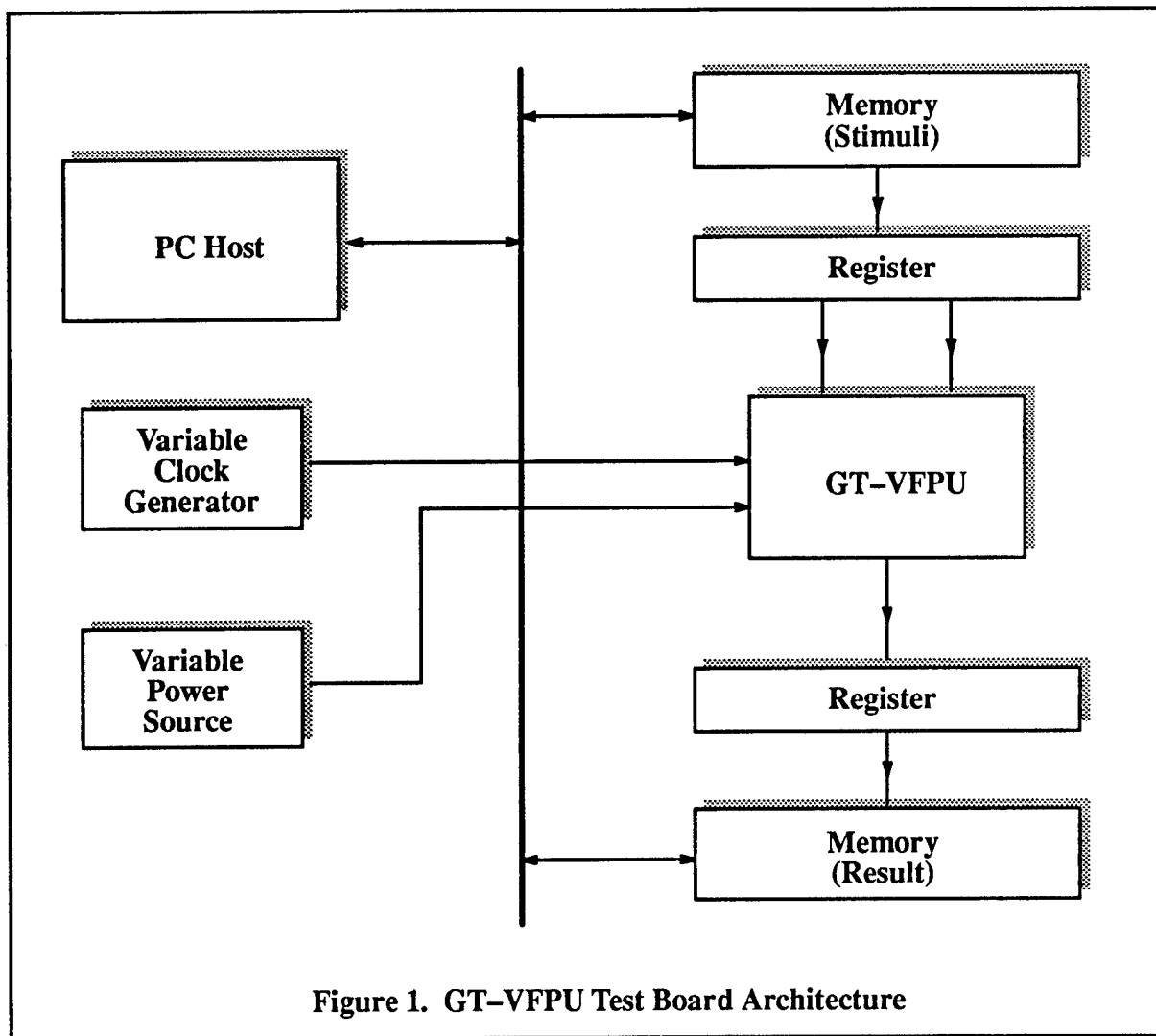
The test software running on the PC Host generates a set of test vectors for each opcode category and downloads them into the memory unit that stores the stimuli for the GT-VFPU. Once instructed to execute, a run-time controller fetches the stimuli sequentially from each of the memory locations. For each stimuli, GT-VFPU computes the result and stores it in a second memory unit. The test software on the PC computes the correct results to be expected and compares this result with the result generated by the GT-VFPU. The memory units are capable of generating and capturing 4096 vectors in a single run.

The test board schematic design is included in Appendix A. The source programs for the on-board GALs are listed in Appendix B. The source code listing of the test software is listed in Appendix C.

3. Test Monitor

A test monitor running on the PC-AT host was developed to control the GT-VFPU test board. It consists of 1,530 lines of Turbo Pascal source code. The available commands are:

```
tmem  : test memory;
tlog  : test xor/and/or/passR/not R/not S;
tiadd : test integer add/sub/rsub;
timult: test integer mult
tfadd : test floating point add
```



tfmult : test floating point mult

tshift : test ROR/ROL/SHR/SHL

tspec0 : test pack exp & float

tspec1 : test seed, unp_exp, unp_man, root_exp, & root_man

tspec2 : test round & trunc

tspec3 : test sign manipulation

tall : test all of the above

dsoe : do not stop on error

soe : stop on error

start : start testing

stop : stop testing

sb : select memory bank

dw : display memory word
sw : substitute memory word
cont : set testing mode to continuous
single : set testing mode to single
debug : toggle debug setting
quit : quit FPU Test Monitor.

Each of the above commands can be invoked from the monitor.

4. Test Strategy

The GT-VFPU opcodes are divided into five broad categories: logical, shift, fixed-point, floating-point, and special. The stimuli for the GT-VFPU consists of the signals R[31:0], S[31:0], and Opcode[4:0]. Two vector sets were used to provide a combination of stimuli for each opcode category.

The first set consists of test vectors generated from a fixed set of patterns devised for each opcode category. Three arrays are used to store the primary test patterns for R[31:0], S[31:0], and Opcode[4:0]. Three, three-level-nested loops are used to generate the test patterns for different combinations of R[31:0], S[31:0], and Opcode[4:0]. Each nested loop places the index of the R[31:0], S[31:0], and Opcode[4:0] arrays in the inner loop. The purpose is to generate a sequential set of patterns that toggle the different sources of input stimuli to the GT-VFPU on a per cycle basis. To insure that the opcode bit fields are toggling every cycle, each odd storage element for the Opcode[4:0] array is stored with an inverted value of the opcode of the preceeding even storage element.

The second vector set consists of random patterns for R[31:0] and S[31:0].

The fixed patterns for each opcode categories are given in the following sections.

4.1. Logical Operations

The logical operations are passR, and, or, xor, not R, and not S. The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$ffffff, \$55555555, \$aaaaaaaa, \$ffffffff, \$00000000, \$12345678, and \$9abcdef0. The pattern \$00000000 is repeated twice to create \$ffffff to \$00000000 and \$00000000 to \$ffffff transistions.

4.2. Shift Operations

The shift operations are shift left, shift right, rotate left, and rotate right. The fixed patterns used for this test for the R[31:0] array are \$00000000, \$ffffff, \$55555555, \$aaaaaaaa, \$ffffffff, \$00000000, \$12345678, and \$9abcdef0. The fixed patterns for the S[31:0] array are \$00000000, \$00000001, \$00000010, ..., \$0000000f.

4.3. Fixed-point Operations

The fixed-point operations are addition, subtraction, reverse subtraction ($-R[31:0] + S[31:0]$), and multiplication. The Multiplication is tested separately.

4.3.1. Addition/Subtraction/Reverse Subtraction

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$00ffffff, \$00555555, \$00aaaaaa, \$00ffffff, \$000000001, \$00123456, \$00abcdef, \$80ffffff, \$80555555, \$80aaaaaa, \$80ffffff, \$800000001, \$80123456, and \$80abcdef.

4.3.2. Multiplication

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$00ffffff, \$00aaaaaa, \$00555555, \$00000fff, \$00000aaa, \$00000555, \$00ffffff, \$80000000, \$80ffffff, \$80aaaaaa, \$80555555, \$80000fff, \$80000aaa, \$80000555, \$80ffffff.

4.4. Floating-Point Operations

The floating-point operations are addition, subtraction, reverse subtraction, and multiplication. The multiplication is tested separately.

4.4.1. Addition/Subtraction/Reverse Subtraction

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$3f000000, \$3ffffff, \$3faaaaa, \$3f555555, \$3f000001, \$7f000000, \$2a800000, \$55000000, \$00800000, \$bf000000, \$bffffff, \$bfaaaaa, \$bf555555, \$bf000001, \$ff000000, \$aa800000, \$d5000000, \$80800000.

4.4.2. Multiplication

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$3ffffff, \$3faaaaa, \$3f555555, \$00000fff, \$00000aaa, \$7f000555, \$7ffffff, \$08000000, \$bffffff, \$bfaaaaa, \$bf555555, \$80000fff, \$80000aaa, \$ff000555, \$ffffff.

4.5. Special Operations

The special operations are pack exponent, float, inverse seed, unpack exponent, unpack mantissa, square root exponent seed, square root mantissa seed, round, trunc, sign of sine, odd to negative, change sign, and sign of tan. The testing of these operations are separated into four groups.

4.5.1. Pack exponent/Float

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00ffffff, \$80000001, \$80000004, \$80000010, \$80000040, \$80000100, \$80ffffff.

4.5.2. Inverse Seed/ Unpack Exponent/ Unpack Mantissa/ Square Root Seed

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$3fffffff, \$00000fff, \$3f000555, \$08000000, \$bfffffff, \$80000fff, \$ff000555.

4.5.3. Round/Trunc

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$3fffffff, \$00000fff, \$3f000555, \$4b000000, \$bfffffff, \$c6000fff, \$bf000555.

4.5.4. Sign of Sine/ Odd to Negative/Change Sign/ Sign of Tan

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are \$00000000, \$00000001, \$80000010, \$80000001, \$3f800000, \$3f800001, \$bf800010, and \$bf800001.

5. Test Result

All Tests were done at room temperature (~75 deg F). Three voltage test conditions were applied to the GT-VFPU chip. The test results are shown in Table 1.

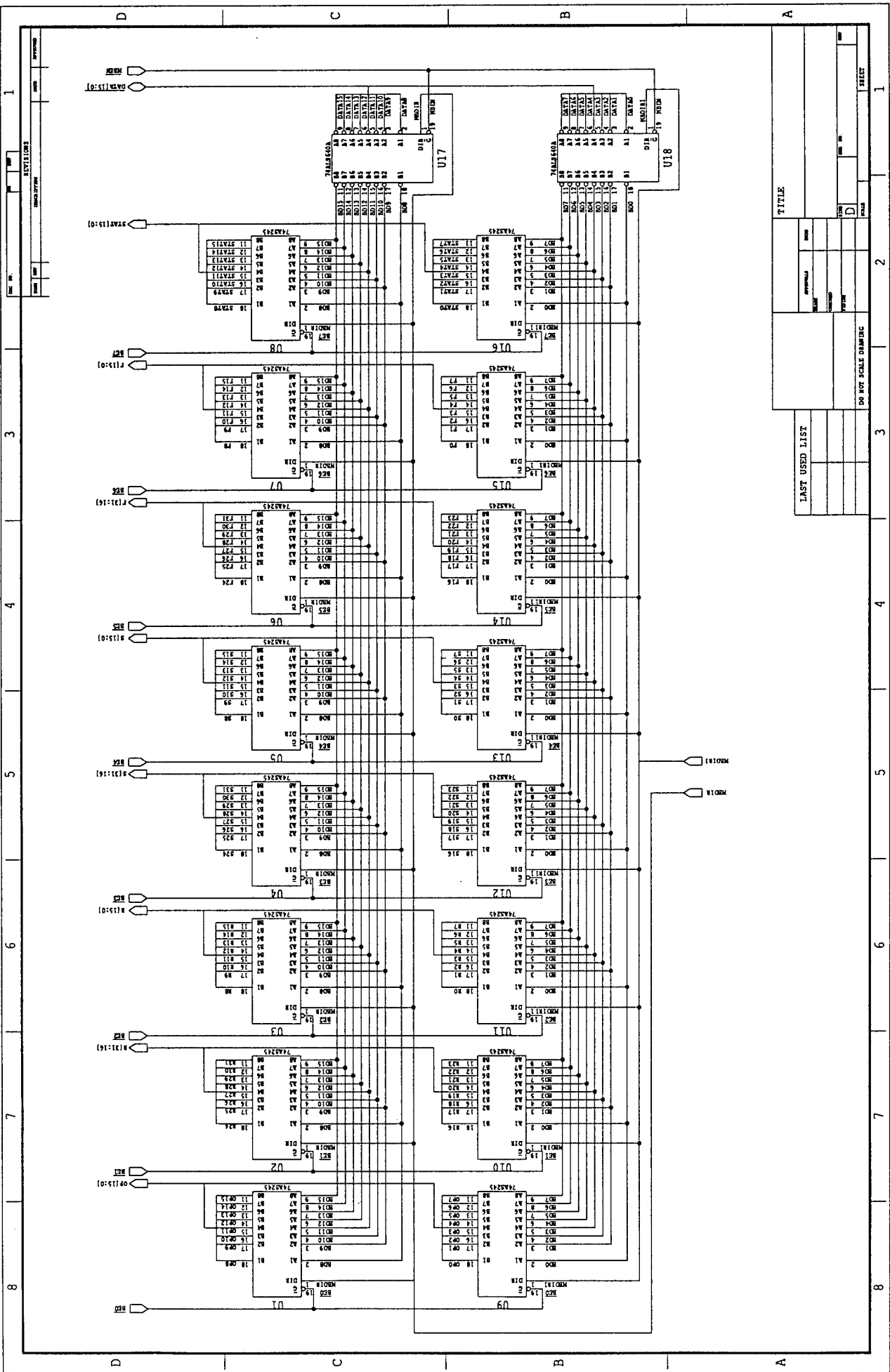
Table 1. Maximum Operating Frequency Characterized by Opcode Category

Opcode Category	4.5 V @ 150 mA	5.0 V @ 190 mA	5.5 V @ 230 mA
logical	19	19	19
Fixed-Point Add	18	19	19
Fixed-Point Mult	19	19	19
Floating-Point Add	16	18	18
Floating-Point Mult	17	19	18
Shift	19	19	18
Special 0	17	18	18
Special 1	17	18	19
Special 2	15	17	19
Special 3	19	19	19
Max/Min	19/15	19/17	19/18

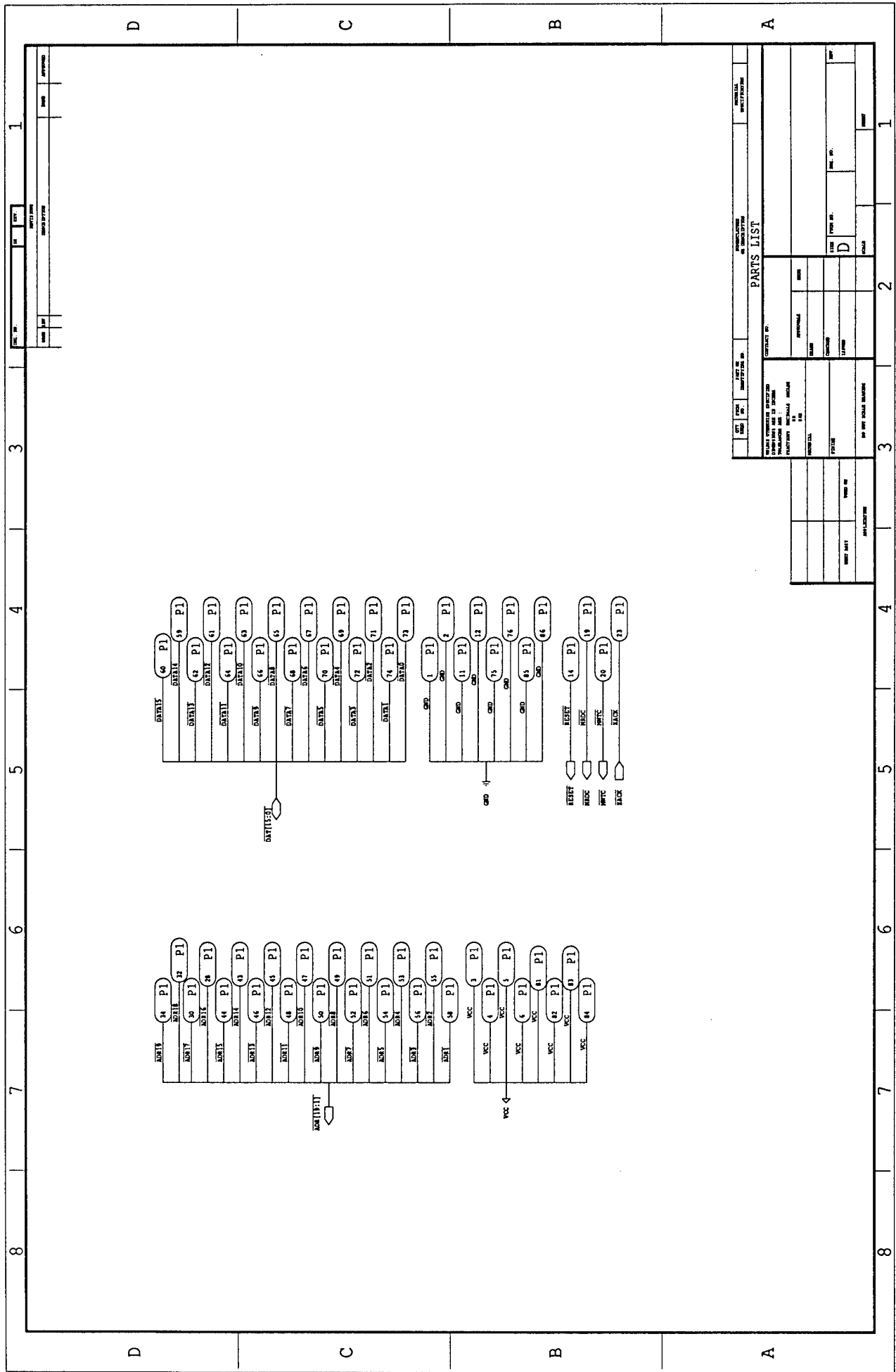
6. Conclusion

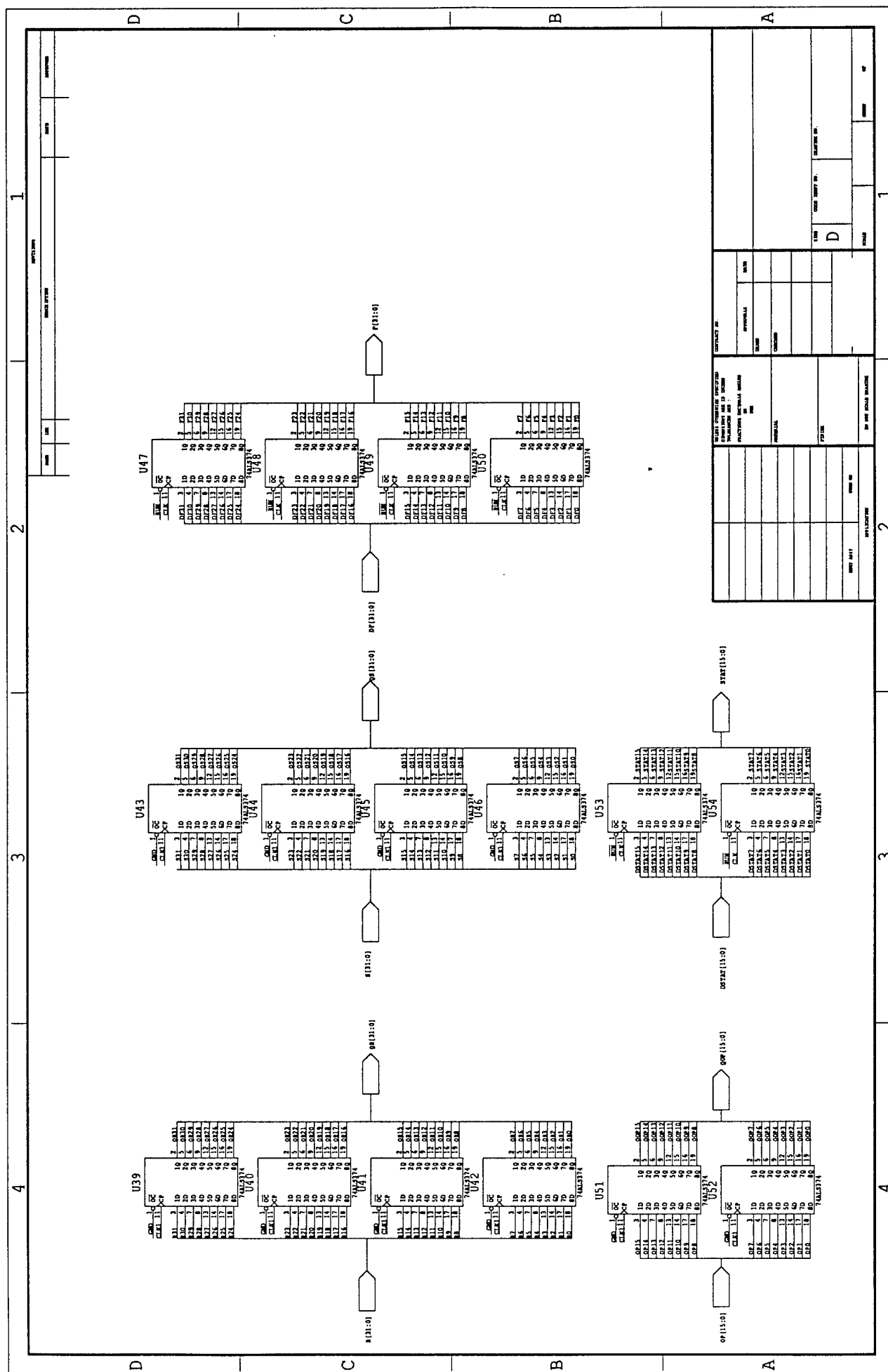
The GT-VFPU test board demonstrated that the Genesil timing and power analysis were very conservative. Genesil predicted a maximum operating frequency of 6.6 Mhz and a power consumption of 5.1 W for the GT-VFPU chips. The test result shows that the GT-VFPU operates at 17 Mhz and consumes only 950 mW with a 5.0 V supply. The highest operating frequency attained is 19 Mhz. This limit may have been imposed by the test board.

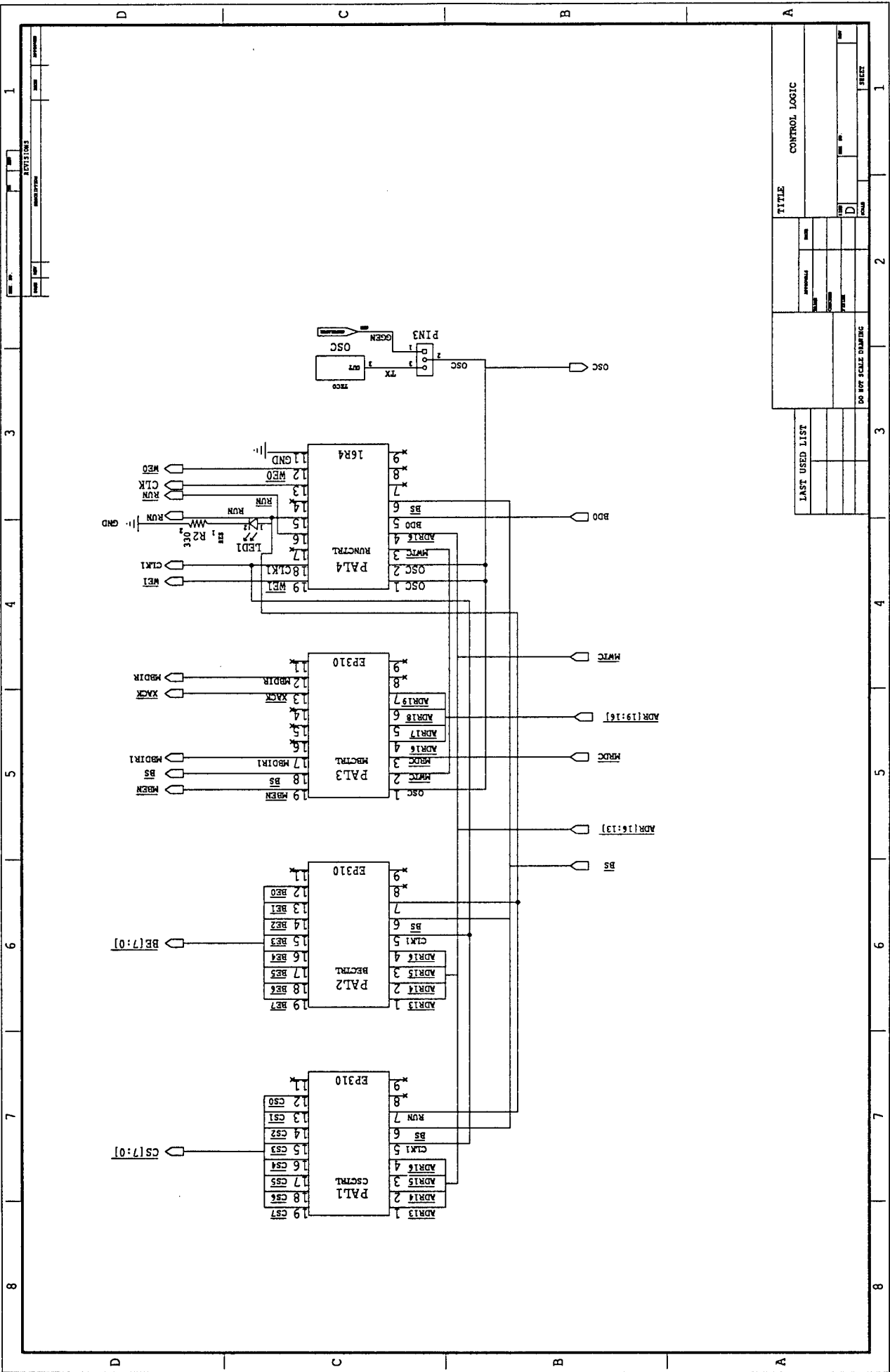
Appendix A
Board Schematics

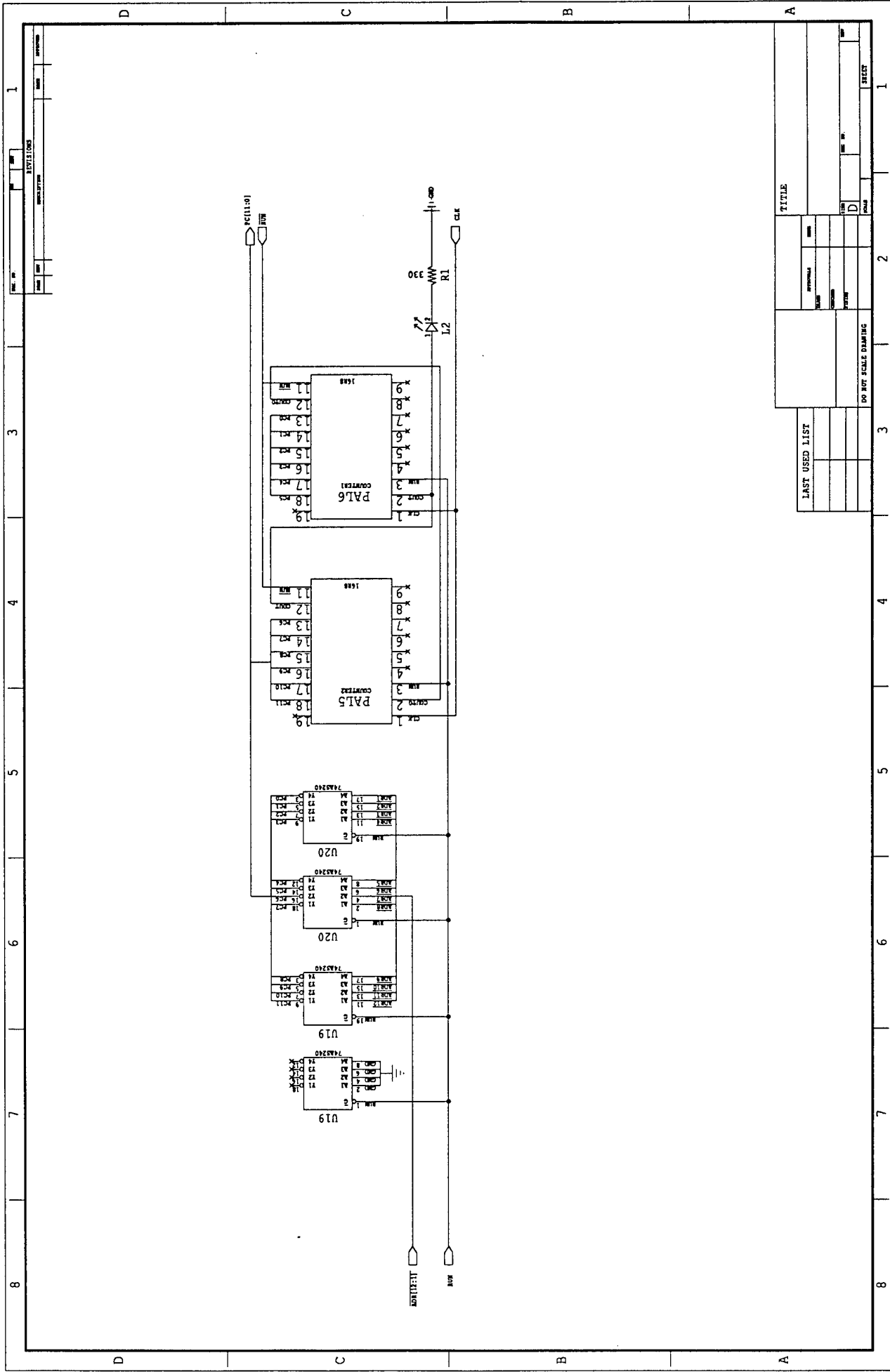


TITLE		DO NOT SCALE DRAWING	
LAST USED LIST			
REV	DATE	BY	CHK
1			
2			
3			
4			
5			
6			
7			
8			

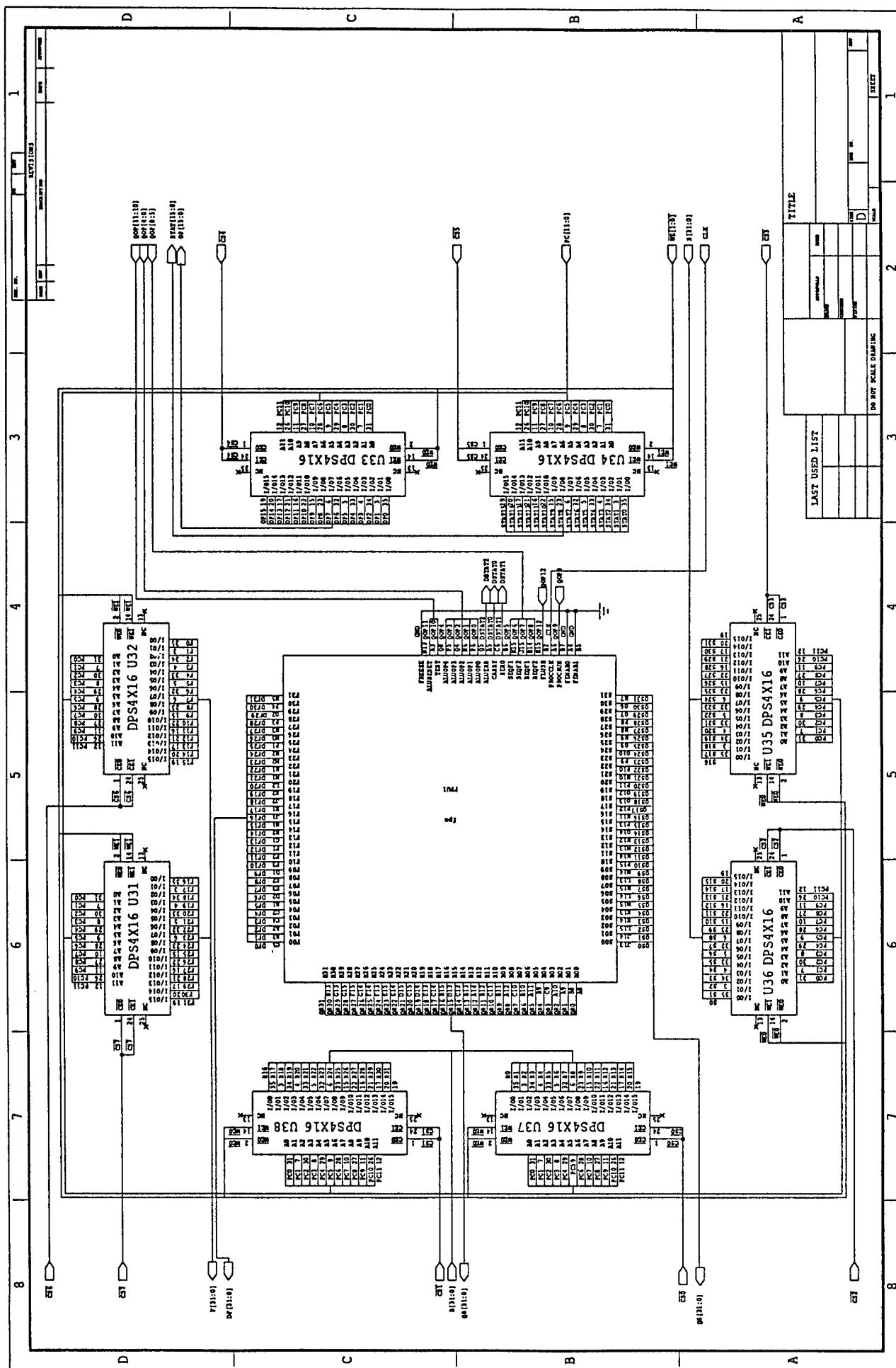








A		A		1		2		3		4		5		6		7		8	
LAST USED LIST		DO NOT SCALE DRAWING		1		2		3		4		5		6		7		8	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY		BY		BY		BY	
DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE		DATE	
BY		BY		BY		BY		BY		BY		BY							



Appendix B

GAL Listing


```

NAME      Runctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal4;
Device    G16V8;

```

```
/* Input */
```

```

Pin 1 = clk_src; /* osc */
Pin 2 = osc;
Pin 3 = !MWTC;
Pin 4 = !ADR_16;
Pin 5 = BD_0;
Pin 6 = !BS;

```

```
/* Output */
```

```

Pin 12 = !we0;
Pin 13 = clk;
Pin 14 = osc_1;
Pin 15 = run;
Pin 16 = n_run;
Pin 17 = osc_2;
Pin 18 = clk1;
Pin 19 = !we1;

```

```
/* Logic Equations */
```

```

run.d    = (BS & MWTC & ADR_16 & BD_0) # ((!BS # !MWTC # !ADR_16) & run);
we0      = MWTC & !run;
we1      = (MWTC & !run) # ((!osc) & run);
osc_1    = osc;
osc_2    = osc_1;
n_run    = !run;
clk      = osc_1;
clk1     = osc_1;

```

```

NAME      Csctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal1;
Device    G16V8; /* EP310 */

```

```
/* Inputs */
```

```

Pin 1 = !ADR_13;
Pin 2 = !ADR_14;
Pin 3 = !ADR_15;
Pin 4 = !ADR_16;
Pin 5 = clk1;
Pin 6 = !BS;
Pin 7 = run;

```

```
/* Outputs */
```

```

Pin 12 = !CS_0;
Pin 13 = !CS_1;
Pin 14 = !CS_2;
Pin 15 = !CS_3;
Pin 16 = !CS_4;
Pin 17 = !CS_5;
Pin 18 = !CS_6;
Pin 19 = !CS_7;

```

/* Logic Equations */

```
CS_7 = BS & !ADR_16 & ADR_15 & ADR_14 & ADR_13 & !run # run;  
CS_6 = BS & !ADR_16 & ADR_15 & ADR_14 & !ADR_13 & !run # run;  
CS_5 = BS & !ADR_16 & ADR_15 & !ADR_14 & ADR_13 & !run # run;  
CS_4 = BS & !ADR_16 & ADR_15 & !ADR_14 & !ADR_13 & !run # run;  
CS_3 = BS & !ADR_16 & !ADR_15 & ADR_14 & ADR_13 & !run # run;  
CS_2 = BS & !ADR_16 & !ADR_15 & ADR_14 & !ADR_13 & !run # run;  
CS_1 = BS & !ADR_16 & !ADR_15 & !ADR_14 & ADR_13 & !run # run;  
CS_0 = BS & !ADR_16 & !ADR_15 & !ADR_14 & !ADR_13 & !run # run;
```

```

Name      Counter1;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company    Cerl;
Assembly  FPU test board;
Location  Pal6;
Device     G16V8;

```

```
/* Inputs */
```

```

Pin 1 = clk;
Pin 2 = cout;
Pin 3 = run;

```

```
/* Output */
```

```

Pin 12 = cout0;
Pin 13 = pc_0;
Pin 14 = pc_1;
Pin 15 = pc_2;
Pin 16 = pc_3;
Pin 17 = pc_4;
Pin 18 = pc_5;

```

```
/* Logic Equations */
```

```

pc_0.d = (!pc_0 # cout) & run;      /* original: cout1 */
pc_1.d = ( pc_0 & !pc_1 # !pc_0 & pc_1 # cout) & run;
pc_2.d = ( pc_0 & pc_1 & !pc_2 # !(pc_0 & pc_1) & pc_2 # cout) & run;
pc_3.d = ( pc_0 & pc_1 & pc_2 & !pc_3 # !(pc_0 & pc_1 & pc_2) & pc_3 # cout) & run;
pc_4.d = ( pc_0 & pc_1 & pc_2 & pc_3 & !pc_4 # !(pc_0 & pc_1 & pc_2 & pc_3) & pc_4 # cout) & run;
pc_5.d = ( pc_0 & pc_1 & pc_2 & pc_3 & pc_4 & !pc_5 # !(pc_0 & pc_1 & pc_2 & pc_3 & pc_4) & pc_5 # cout) & run;

```

```
cout0 = pc_0 & pc_1 & pc_2 & pc_3 & pc_4 & pc_5;
```

```

Name      Counter2;
Partno    000;
Date      09/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company    Cerl;
Assembly  FPU test board;
Location  PAL5;
Device     G16V8;

```

```
/* Input */
```

```

Pin 1 = clk;
Pin 2 = cout0;
Pin 3 = run;

```

```
/* Output */
```

```

Pin 12 = cout;
Pin 13 = pc_6;
Pin 14 = pc_7;
Pin 15 = pc_8;
Pin 16 = pc_9;
Pin 17 = pc_10;
Pin 18 = pc_11;

```

```
/* Logic Equations */
```

```

pc_6.d = (cout0 & !pc_6 # !cout0 & pc_6 # cout) & run;
pc_7.d = (cout0 & pc_6 & !pc_7 # !(cout0 & pc_6) & pc_7 # cout) & run;
pc_8.d = (cout0 & pc_7 & pc_6 & !pc_8 # !(cout0 & pc_7 & pc_6) & pc_8 # cout) & run;

```

```

pc_9.d = (cout0 & pc_8 & pc_7 & pc_6 & !pc_9 #
          !(cout0 & pc_8 & pc_7 & pc_6) & pc_9 # cout) & run;
pc_10.d = (cout0 & pc_9 & pc_8 & pc_7 & pc_6 & !pc_10 #
           !(cout0 & pc_9 & pc_8 & pc_7 & pc_6) & pc_10 # cout) & run;
pc_11.d = (cout0 & pc_10 & pc_9 & pc_8 & pc_7 & pc_6 & !pc_11 #
           !(cout0 & pc_10 & pc_9 & pc_8 & pc_7 & pc_6) & pc_11 # cout) &
          run;
cout    = cout0 & pc_11 & pc_10 & pc_9 & pc_8 & pc_7 & pc_6;

```

```

NAME      Bectrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal2;
Device    G16V8;

```

```
/* Input */
```

```

Pin 1 = !ADR_13;
Pin 2 = !ADR_14;
Pin 3 = !ADR_15;
Pin 4 = !ADR_16;
Pin 5 = clk1;
Pin 6 = !BS;
Pin 7 = run;

```

```
/* Output */
```

```

Pin 12 = !Be_0;
Pin 13 = !Be_1;
Pin 14 = !Be_2;
Pin 15 = !Be_3;
Pin 16 = !Be_4;
Pin 17 = !Be_5;
Pin 18 = !Be_6;
Pin 19 = !Be_7;

```

```
/* Logic Equations */
```

```

Be_7 = BS & !ADR_16 & ADR_15 & !ADR_14 & ADR_13 & !run;
Be_6 = BS & !ADR_16 & ADR_15 & ADR_14 & !ADR_13 & !run;
Be_5 = BS & !ADR_16 & ADR_15 & ADR_14 & ADR_13 & !run;
Be_4 = BS & !ADR_16 & !ADR_15 & ADR_14 & !ADR_13 & !run;
Be_3 = BS & !ADR_16 & !ADR_15 & ADR_14 & ADR_13 & !run;
Be_2 = BS & !ADR_16 & !ADR_15 & !ADR_14 & !ADR_13 & !run;
Be_1 = BS & !ADR_16 & !ADR_15 & !ADR_14 & ADR_13 & !run;
Be_0 = BS & !ADR_16 & ADR_15 & !ADR_14 & !ADR_13 & !run;

```

```

NAME      Mbctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company    Cerl;
Assembly  FPU test board;
Location  Pal3;
Device    G16V8;

```

```
/* Input */
```

```

Pin 1 = osc;
Pin 2 = !MWTC;
Pin 3 = !MRDC;
Pin 4 = !ADR_16;
Pin 5 = !ADR_17;
Pin 6 = !ADR_18;
Pin 7 = !ADR_19;

```

```
/* Output */
```

```

Pin 12 = MBdir;
Pin 13 = !Xack;
Pin 14 = Xack1;
Pin 15 = Xack0;
Pin 17 = MBdir1;
Pin 18 = !BS;
Pin 19 = !MBen;

```

```
/* Logic Equations */
```

```

Xack0.D = !Xack & Xack1 & (MWTC # MRDC);
Xack1.D = !Xack & !Xack0 & (MWTC # MRDC) #
          Xack1 & Xack0 & (MWTC # MRDC);

```

```
/* if BS */
```

```
Xack.OE = BS;
```

```

Xack.D = (Xack1 & Xack0 & (MWTC # MRDC) #
          Xack & !Xack0 & (MWTC # MRDC));

```

```
MBen = MWTC # MRDC;
```

```
MBdir = MWTC;
```

```
MBdir1 = MWTC;
```

```
BS = !ADR_19 & !ADR_18 & !ADR_17;
```

```

NAME      Runctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company    Cerl;
Assembly  FPU test board;
Location  Pal4;
Device    G16V8;

```

```
/* Input */
```

```

Pin 1 = clk_src; /* osc */
Pin 2 = osc;
Pin 3 = !MWTC;
Pin 4 = !ADR_16;
Pin 5 = BD_0;
Pin 6 = !BS;

```

```
/* Output */
```

```

Pin 12 = !we0;
Pin 13 = clk;
Pin 14 = osc_1;
Pin 15 = run;
Pin 16 = n_run;
Pin 17 = osc_2;
Pin 18 = clk1;
Pin 19 = !we1;

```

```
/* Logic Equations */
```

```

run.d    = (BS & MWTC & ADR_16 & BD_0) # ((!BS # !MWTC # !ADR_16) & run);
we0      = MWTC & !run;
we1      = (MWTC & !run) # ((!osc) & run);
osc_1    = osc;
osc_2    = osc_1;
n_run    = !run;
clk      = osc_1;
clk1     = osc_1;

```

Appendix C
Test Monitor Source Code


```

program testfpu;
uses ieee_cnv,
    {$u e:\dp\dp_comp\hex_conv} hex_conv,
    io,dos;

const
    max_test_vector = 4092;
    ( logical )
    fpu_and = $8;
    fpu_or = $9;
    fpu_xor = $a;
    fpu_notR = $b;
    fpu_notS = $c;
    fpu_passR = $14;
    ( fixed )
    fpu_add = $0;
    fpu_sub = $1;
    fpu_mult = $2;
    fpu_rsub = $3;
    ( float )
    fpu_fadd = $10;
    fpu_fsub = $11;
    fpu_fmuilt = $12;
    fpu_frsub = $13;
    ( shift )
    fpu_ror = $4;
    fpu_rol = $5;
    fpu_shr = $6;
    fpu_shl = $7;
    ( special )
    fpu_pack = $0d;
    fpu_float = $15;
    fpu_seed = $e;
    fpu_unp_exp = $18;
    fpu_unp_man = $19;
    fpu_rootexp = $1a;
    fpu_rootman = $1b;
    fpu_round = $f;
    fpu_int_r = $16;
    fpu_int_s = $17;
    fpu_sin_sgn = $1c;
    fpu_odd_neg = $1d;
    fpu_chg_sgn = $1e;
    fpu_tan_sgn = $1f;

    ( test cases )
    logical = 0;
    iadd = 1;
    imult = 2;
    fadd = 3;
    fmuilt = 4;
    shift = 5;
    special = 6;

var bank,offset,i,address : word;
    start_bank, end_bank : word;
    pattern : array[0..20] of word;
    ch : char;
    command : string;
    debug : boolean;
    rsign : integer;
    ssign : integer;
    test_case : integer;
    last_op,last_s,last_r : integer;
    r,s : array[0..4095] of longint;
    op : array[0..31] of word;
    cflag, zflag : integer;
    continuous : char;
    stop_on_error : char;
    count : integer;
    test_cycle : longint;
    no_error : longint;

procedure write_error(procedure_name,message:string);
begin
    writeln('Error at procedure ',procedure_name,' !!!');
    writeln(' ',message);
    halt;
end;

```

```

procedure mwrite(bank,offset,pattern:word);
begin
    memw[segment:(bank shl 13) + offset] := pattern;
end;

function mread(bank,offset:word):word;
begin
    mread := memw[segment:(bank shl 13) + offset];
end;

procedure verify(bank,offset,data,expdata:word);
begin
    if data <> expdata then
        begin
            writeln('error at bank ',bank,' at location ',word_to_hex(offset));
            writeln('  written: ',word_to_hex(expdata));
            write ('  read   : ',word_to_hex(data),' <CR> '); readln;
            writeln;
        end;
    end;
end;

procedure test_memory;
begin
    stop_processor;
    pattern[0] := $1234;
    pattern[1] := $0000;
    pattern[2] := $5555;
    pattern[3] := $aaaa;
    pattern[4] := $ffff;
    pattern[5] := $ff00;
    pattern[6] := $00ff;
    write('test all banks ? '); readln(ch);
    if (ch = 'y') or (ch = 'Y') then
        begin
            start_bank := 0; end_bank := 7;
        end
    else
        begin
            write('Which bank to test ? '); readln(start_bank);
            end_bank := start_bank;
        end;
    begin
        for bank := start_bank to end_bank do
            begin
                for i := 0 to 6 do
                    begin
                        address := 0;
                        writeln('testing pattern ',word_to_hex(pattern[i]),' on bank ',bank);
                        while address <= $1ffe do
                            begin
                                mwrite(bank,address,pattern[i]);
                                verify(bank,address,mread(bank,address),pattern[i]);
                                address := address + 2;
                            end;
                        end;
                        address := 0;
                        writeln('writing address on bank ',bank);
                        while address <= $1ffe do
                            begin
                                mwrite(bank,address,address);
                                address := address + 2;
                            end;
                        address := 0;
                        writeln('reading address on bank ',bank);
                        while address <= $1ffe do
                            begin
                                verify(bank,address,mread(bank,address),address);
                                address := address + 2;
                            end;
                        end;
                    end;
                if (ch='y') or (ch='Y') then
                    begin
                        address := 0;
                        writeln('writing address to all banks');
                        repeat
                            mwrite(0,address,address);
                            address := address + 2;
                        until address = $fffe;
                        mwrite(0,address,address);
                        address := 0;
                        writeln('reading address from all banks');
                        repeat

```

```

        verify(0,address,mread(0,address),address);
        address := address + 2;
        until address = $fffe;
        verify(0,address,mread(0,address),address);
    end;
    if (ch <> 'y') and (ch <> 'Y') then exit;
end;
writeln('memory testing completed');
end; { of test_memory }

procedure write_fpu_vector(address:word;r,s:longint;op:word);
var lsw,msw : longint;
begin
    address := address shl 1;
    lsw := $0000ffff and r;
    msw := r shr 16;
    mwrite(0,address,lsw);
    mwrite(1,address,msw);
    verify(0,address,mread(0,address),lsw);
    verify(1,address,mread(1,address),msw);
    lsw := $0000ffff and s;
    msw := s shr 16;
    mwrite(2,address,lsw);
    mwrite(3,address,msw);
    verify(2,address,mread(2,address),lsw);
    verify(3,address,mread(3,address),msw);
    op := op or $0200; { set proc_run to 1 }
    mwrite(4,address,op);
    verify(4,address,mread(4,address),op);
end; { of write_fpu_vector }

procedure check_fpu_vector(address:word;r,s:longint;op:word);
var lsw,msw : longint;
begin
    address := address shl 1;
    lsw := $0000ffff and r;
    msw := r shr 16;
    verify(0,address,mread(0,address),lsw);
    verify(1,address,mread(1,address),msw);
    lsw := $0000ffff and s;
    msw := s shr 16;
    verify(2,address,mread(2,address),lsw);
    verify(3,address,mread(3,address),msw);
    op := op or $0200; { set proc_run to 1 }
    verify(4,address,mread(4,address),op);
    mwrite(6,address+8,0);
    mwrite(7,address+8,0);
    verify(6,address,mread(6,address+8),0);
    verify(7,address,mread(7,address+8),0);
end; { of write_fpu_vector }

function sm2twosc(d:longint):longint;
begin
    if (d and $80000000) <> 0 then
        d := -(d and $00ffffff)
    else
        d := d and $00ffffff;
    sm2twosc := d;
end;

function twosc2sm(d:longint):longint;
begin
    if (d < 0) then
        d := (-d and $7fffffff) or $80000000;
    if (d and $01000000) <> 0 then cflag := 1;
    twosc2sm := d and $80ffffff;
end;

function compute_fadd(r,s:longint):longint;
var
    addsubsel,expdiff,bgta,explarge : longint;
    output,mantlarge,intermediate,i : longint;
    sexp,rexp,smant,rmant,ssign,rsign : longint;
    overflow,underflow : longint;
begin
    sexp := (s and $7f800000) shr 23;
    rexp := (r and $7f800000) shr 23;
    smant := s and $007fffff;
    if (s<>0) then smant := smant or $00800000;
    rmant := r and $007fffff;
    if (r<>0) then rmant := rmant or $00800000;

```

```

ssign := (s and $80000000) shr 31;
rsign := (r and $80000000) shr 31;

addsubsel := (1 xor ssign xor rsign);
expdiff := rexp - sexp;
bgta := 0;
explarge := rexp;
output := rsign;

if (expdiff < 0) then
begin
    expdiff := -expdiff;
    bgta := 1;
    explarge := sexp;
    output := ssign;
end;

explarge := explarge + 1;

if (bgta = 0) then
begin
    if (expdiff >= 1) then
        intermediate := smant shr (expdiff-1)
    else
        intermediate := smant shl 1;
    mantlarge := rmant shl 1;
end
else
begin
    if (expdiff >= 1) then
        intermediate := rmant shr (expdiff-1)
    else
        intermediate := rmant shl 1;
    intermediate := rmant shr (expdiff-1);
    mantlarge := smant shl 1;
end;

if (expdiff > 23) then intermediate := 0;

if (addsubsel = 1) then
    intermediate := mantlarge + intermediate
else
    intermediate := mantlarge - intermediate;

if (intermediate < 0) then
begin
    intermediate := -intermediate;
    output := output xor 1;
end;
intermediate := intermediate shr 1;

(writeln('intermed ', longint_to_hex(intermediate)));

i := 0;
while ( ((intermediate and $01000000)=0) and (i<25) ) do
begin
    intermediate := intermediate shl 1;
    explarge := explarge - 1;
    i := i + 1;
end;
intermediate := intermediate shr 1;

overflow := (explarge and $100) shr 8;
underflow := (explarge and $80) shr 7;

if ( (underflow and overflow) <> 0 ) then
begin
    output := 0;
    zflag := 1;
    overflow := 0;
end
else
begin
    output := output shl 31;
    output := output or ( (explarge and $ff) shl 23 );
    output := output or ( intermediate and $7ffff );
end;

if (intermediate = 0) then
begin
    output := 0;
    zflag := 1;
    overflow := 0;
end;

```

```

        cflag := 0;
    (
        if (output <> 0) then
            writeln('fmant', longint_to_hex( (output and $7ffffff) or $8000000 ))
        else
            writeln('fmant', longint_to_hex( 0 ));
        writeln('f      ', longint_to_hex(output));
        writeln('fexp ', longint_to_hex( (output and $7f800000) shr 23 ));
    )

    compute_fadd := output;
end; ( of compute_fadd )

function compute_fmulo(r,s:longint):longint;
var
    rmanthi,rmantlo,smanthi,smantlo : longint;
    output,w,x,y,z,reshi,reslo,intermediate : longint;
    sexp,rexp,smant,rmant,ssign,rsign : longint;
    overflow,underflow : longint;
begin
    sexp := (s and $7f800000) shr 23;
    rexp := (r and $7f800000) shr 23;
    smant := s and $007fffff;
    if (s<>0) then smant := smant or $00800000;
    rmant := r and $007fffff;
    if (r<>0) then rmant := rmant or $00800000;
    ssign := (s and $80000000) shr 31;
    rsign := (r and $80000000) shr 31;

    rmantlo := rmant and $ffff;
    rmanthi := rmant shr 16;
    smantlo := smant and $ffff;
    smanthi := smant shr 16;
    w := rmantlo * smantlo;
    x := rmanthi * smanthi;
    y := rmantlo * smanthi;
    z := rmanthi * smanthi;
    intermediate := ( (w shr 16) and $ffff ) + x + y;
    reslo := ( (intermediate shl 16) and $ffff0000 ) or (w and $ffff);
    reshi := ( (intermediate shr 16) and $ffff ) + z;

    (
        writeln('rmant ', longint_to_hex(rmant));
        writeln('smant ', longint_to_hex(smant));
        writeln('reshi ', longint_to_hex(reshi));
        writeln('reslo ', longint_to_hex(reslo));
    )

    intermediate := rexp + sexp;
    if ( (reshi and $8000) <> 0 ) then
        begin
            output := (reshi shl 8) or ( (reslo shr 24) and $ff );
            intermediate := intermediate - 126;
        end
    else
        begin
            output := (reshi shl 9) or ( (reslo shr 23) and $1ff );
            intermediate := intermediate - 127;
        end;
    end;

    overflow := (intermediate and $100) shr 8;
    cflag := 0;
    zflag := 0;
    underflow := ( ( (not rexp) and (not sexp) ) shr 7 ) and 1;
    if ( (overflow and underflow) <> 0 ) then
        begin
            output := 0;
            zflag := 1;
            overflow := 0;
        end
    else
        begin
            output := output and $7ffffff;
            output := output or ( (ssign xor rsign) shl 31);
            output := output or ( (intermediate and $ff) shl 23 );
        end;
    end;

    if ( (reshi or reslo) = 0 ) then
        begin
            output := 0;
            zflag := 1;
            overflow := 0;
        end;
    end;
end;

```

```

        compute_fmuilt := output;
end;{ of compute_fmuilt }
function compute_float(r:longint):longint;
var
    explarge : longint;
    output,intermediate,i : longint;
    rexp,rmant,rsign : longint;
    overflow,underflow : longint;
begin
    rmant := r and $00ffffff;
    rsign := (r and $80000000) shr 31;
    explarge := $97;
    intermediate := rmant;
    i := 0;
    while ( ((intermediate and $01000000)=0) and (i<25) ) do
        begin
            intermediate := intermediate shl 1;
            explarge := explarge - 1;
            i := i + 1;
        end;
    intermediate := intermediate shr 1;

    overflow := (explarge and $100) shr 8;
    underflow := (explarge and $80 ) shr 7;
    if ( (underflow and overflow) <> 0 ) then
        begin
            output := 0;
            zflag := 1;
            overflow := 0;
        end
    else
        begin
            output := rsign shl 31;
            output := output or ( (explarge and $ff) shl 23 );
            output := output or ( intermediate and $7fffff );
        end;

    if (intermediate = 0) then
        begin
            output := 0;
            zflag := 1;
            overflow := 0;
        end;

    cflag := 0;
    compute_float := output;
end;{ of compute_float }
function compute_unp_exp(r:longint):longint;
var
    explarge : longint;
    output,intermediate,i : longint;
    rexp,rmant,rsign : longint;
    overflow,underflow : longint;
begin
    rexp := (r shr 23) and $ff;
    rsign := 0;
    rexp := rexp - 127;
    if (rexp < 0) then
        begin
            rexp := -rexp;
            rsign := 1;
        end;

    explarge := $97;
    intermediate := rexp;
    i := 0;
    while ( ((intermediate and $01000000)=0) and (i<25) ) do
        begin
            intermediate := intermediate shl 1;
            explarge := explarge - 1;
            i := i + 1;
        end;
    intermediate := intermediate shr 1;

    output := rsign shl 31;
    output := output or ( (explarge and $ff) shl 23 );
    output := output or ( intermediate and $7fffff );
    zflag := 0;

```

```

        if (intermediate = 0) then
        begin
            output := 0;
            zflag := 1;
            overflow := 0;
        end;

        cflag := 0;
        compute_unp_exp := output;
    end;{ of compute_unp_exp }

function compute_int(r:longint;round:integer):longint;
var
    expdiff,bgta,reshi : longint;
    output,intermediate,carryin : longint;
    rexp,rmant,rsign : longint;
    overflow,underflow : longint;
begin
    rexp := (r and $7f800000) shr 23;
    rmant := r and $007fffff;
    if (r<>0) then rmant := rmant or $00800000;
    rsign := (r and $80000000) shr 31;

    expdiff := $00000096 - rexp;
    reshi := $96;
    intermediate := rmant;
    bgta := 1;
    if (expdiff < 0) then
    begin
        reshi := rexp;
        expdiff := -expdiff;
        bgta := 0;
    end;

    if (expdiff > 0) then
        carryin := ( intermediate shr (expdiff - 1) ) and 1
    else
        carryin := 0;

    intermediate := intermediate shr (expdiff * bgta);
    if ( (round=1) and (bgta=1) ) then intermediate := intermediate + carryin;
    if ( (bgta=1) and (expdiff>24) ) then intermediate := 0;
    reshi := reshi + 1;

    overflow := 0;
    cflag := 0;
    zflag := 0;
    if (reshi <> $97) then overflow := 1;
    if (intermediate = 0) then zflag := 1;

    if ( zflag=1 ) then
        output := 0
    else
    begin
        output := rsign shl 31;
        output := output or ( intermediate and $ffffff );
    end;

    compute_int := output;
end;{ of compute_int }

procedure check_fpu_result(address:word;r,s:longint;op:word);
var f,msw,lsw,readf: longint;
    tr,ts : longint;
    rsign,ssign : longint;
    sr,ss : single;
    readcflag,readzflag : integer;
    i : integer;
    sexp,rexp,fexp,smant,rmant,fmant : longint;
    Year,Month,Day,DayOfWeek : word;
    Hour,Minute,Second,Sec100 : word;
begin
    cflag := 0;
    zflag := 0;
    address := (address+4) shl 1;
    case op of
        fpu_and: f := r and s;
        fpu_or : f := r or s;
        fpu_xor : f := r xor s;
        fpu_notR: f := not r;
        fpu_notS: f := not s;
    end;

```

```

fpw_passR: f := r;
fpw_add : f := twosc2sm(sm2twosc(r) + sm2twosc(s));
fpw_sub : f := twosc2sm(sm2twosc(r) - sm2twosc(s));
fpw_rsub : f := twosc2sm(sm2twosc(s) - sm2twosc(r));
fpw_mult :
begin
  ssign := s and $80000000;
  rsign := r and $80000000;
  tr := r and $ffffff;
  ts := s and $ffffff;
  f := 0;
  for i := 0 to 23 do
  begin
    if (ts and 1) = 1 then f := f + tr;
    tr := tr shl 1; ts := ts shr 1;
  end;
  if (f and $01000000) <> 0 then cflag := 1;
  if f = 0 then
  begin
    rsign := 0;
    ssign := 0;
  end;
  f := (rsign xor ssign) or (f and $00ffffff);
end;
{ float }
fpw_fadd : f := compute_fadd(r,s);
fpw_fsub : f := compute_fadd(r,(s xor $80000000));
fpw_fmull : f := compute_fmull(r,s);
fpw_frsub : f := compute_fadd((r xor $80000000),s);
fpw_ror : f := ((r shl (32 - (s and $1f))) or (r shr (s and $1f)));
fpw_rol : f := ((r shr (32 - (s and $1f))) or (r shl (s and $1f)));
fpw_shr : f := r shr (s and $1f);
fpw_shl : f := r shl (s and $1f);
fpw_pack :
begin
  f := r and $ffffff;
  if ( (r and $80000000)=0 ) then
    f := 127 + f
  else
    f := 127 - f;
  cflag := 0;
  zflag := 0;
  if ( (f < 0) or ((f and $ff)=0) ) then
  begin
    f := 0;
    zflag := 1;
  end;
  f := ( (f shl 23) and $7f800000 );
end;
fpw_float : f := compute_float(r);
fpw_seed :
begin
  f := (r and $7f800000) shr 23;
  f := f + 2;
  cflag := 0;
  zflag := 0;
  if ( (r and $7fffffff)=0 ) then
  begin
    zflag := 1;
    f := 0;
  end
  else
  begin
    f := (r and $80000000) or ( (f and $ff) shl 23 ) or $400000;
    f := f xor $7f800000;
  end;
end;
fpw_unp_exp : f := compute_unp_exp(r);
fpw_unp_man :
begin
  f := (s and $807fffff) or $3f800000;
  zflag := 0;
  cflag := 0;
  if ( (s and $7fffffff) = 0 ) then
  begin
    zflag := 1;
    f := 0;
  end;
end;
fpw_rootexp :
begin

```



```

if ((r and $80000000)=0) then
  if ( (r and $00800000) <> 0 ) then
    f := ( (r shr 24) and $7f ) + 64
  else
    f := ( (r shr 24) and $7f ) + 63
  else
    if ( (r and $00800000) <> 0 ) then
      f := -( (r shr 24) and $7f ) + 64
    else
      f := -( (r shr 24) and $7f ) + 63;
    if (f<0) then f := -f;
    f := (f and $ff) shl 23;
    cflag := 0;
    zflag := 0;
    if ((r and $7fffffff)=0) then
      begin
        zflag := 1;
        f := 0;
      end;
    end;
  end;
fpu_rootman :
begin
  if ( (s and $00800000) = 0) then
    f := (s and $007fffff) or $40000000
  else
    f := (s and $007fffff) or $3f800000;
    cflag := 0;
    zflag := 0;
    if ((s and $7fffffff)=0) then
      begin
        zflag := 1;
        f := 0;
      end;
    end;
  end;
fpu_round : f:= compute_int(r,1);
fpu_int_r : f:= compute_int(r,0);
fpu_int_s : f:= compute_int(s,0);
fpu_sin_sgn :
begin
  f := (s xor (r shl 31)) and $80000000;
  f := f or (s and $7fffffff);
  zflag := 0;
  cflag := 0;
  if ((f and $7fffffff)=0) then
    begin
      zflag := 1;
      f := 0;
    end;
  end;
end;
fpu_odd_neg :
begin
  f := (s shl 31) or (s and $ffffff);
  zflag := 0;
  cflag := 0;
  if ((s and $ffffff)=0) then
    begin
      zflag := 1;
      f := 0;
    end;
  end;
end;
fpu_chg_sgn :
begin
  f := (r and $80000000) or (s and $7fffffff);
  zflag := 0;
  cflag := 0;
  if ((s and $7fffffff)=0) then
    begin
      zflag := 1;
      f := 0;
    end;
  end;
end;
fpu_tan_sgn :
begin
  f := (r shr 31) xor (s shr 31) xor r;
  f := f shl 31;
  f := f or (s and $7fffffff);
  cflag := 0;
  zflag := 0;
  if ((f and $7fffffff)=0) then
    begin
      zflag := 1;
    end;
  end;
end;

```

```

        f := 0;
    end;
end;
end; { of opcode case }
if (f=0) then zflag := 1 else zflag := 0;
lsb := mread(6,address);
msw := mread(7,address);
readf := (msw shl 16) or lsb;
lsb := mread(5,address);
readcflag := (lsb and 1);
readzflag := (lsb and 2) shr 1;
if (readf <> f) or (readcflag <> cflag) or (readzflag <> zflag) or (debug) then
begin
    if (readf <> f) or (readcflag <> cflag) then no_error := no_error + 1;
    write(word_to_hex(address shr 2),':',longint_to_hex(r),' ');
    case op of
        fpu_and : write('and ');
        fpu_or  : write('or ');
        fpu_xor  : write('xor ');
        fpu_add  : write('fix+ ');
        fpu_sub  : write('fix- ');
        fpu_mult : write('fix* ');
        fpu_rsub : write('fixr-');
        fpu_passr: write('passR');
        fpu_pack : write('pack exp R');
        fpu_float : write('float R');
        fpu_seed  : write('inv seed R');
        fpu_unp_exp : write('unpack exp R');
        fpu_unp_man : write('unpack mant S');
        fpu_rootexp : write('root exp R');
        fpu_rootman : write('root mant S');
        fpu_round  : write('round R');
        fpu_int_r  : write('truncate R');
        fpu_int_s  : write('truncate S');
        fpu_sin_sgn : write('sine sign');
        fpu_odd_neg : write('odd negative S');
        fpu_chg_sgn : write('change sign');
        fpu_tan_sgn : write('tangent sign');
        fpu_fadd  : write('float+ ');
        fpu_fsub  : write('float- ');
        fpu_fmuls : write('float* ');
        fpu_frsub : write('floatr-');
    end;
    write(' ',longint_to_hex(s),' -> ');
    write(longint_to_hex(f),'[',zflag,',',cflag,'] (pc) ');
    write(longint_to_hex(readf),'[',readzflag,',',readcflag,'] (fpu) ');
    if (readf <> f) or (readcflag <> cflag) then
    begin
        GetTime(Hour,Minute,Second,Sec100);
        GetDate(Year,Month,Day,DayOfWeek);
        write('Date: ',Month,'/',Day,'/',Year,' at ',Hour,':',Minute,':',Second,' ');
        if (command = 'tall') then writeln('Error at cycle ',count,'.')
        else writeln('Error at cycle ',test_cycle,'.');
    end;
    if (stop_on_error = 'y') then readln else writeln;
end; { of readf <> f }
end; { of check_fpu_result }

procedure generate_vectors(phase : integer);
var i,j,k : integer;
begin
    address := 2;
    writeln('generating test vectors phase ',phase);
    case phase of
        0:
            for k := 0 to last_op do
                for j := 0 to last_s do
                    for i := 0 to last_r do
                        begin
                            if address >= max_test_vector then write_error('','too many test vectors');
                            write_fpu_vector(address,r[i],s[j],op[k]);
                            address := address + 1;
                        end;
                    end;
                end;
            end;
        1:
            for k := 0 to last_op do
                for i := 0 to last_r do
                    for j := 0 to last_s do
                        begin
                            if address >= max_test_vector then write_error('','too many test vectors');
                            write_fpu_vector(address,r[i],s[j],op[k]);
                            address := address + 1;
                        end;
                    end;
                end;
            end;
    end;
end;

```

```

        end;
2:   for j := 0 to last_s do
        for i := 0 to last_r do
            for k := 0 to last_op do
                begin
                    if address >= max_test_vector then write_error('','too many test vectors');
                    write_fpu_vector(address,r[i],s[j],op[k]);
                    address := address + 1;
                end;
            end;
        end; { of generate_vectors }
    procedure wait;
    var i : integer;
    begin
        for i := 0 to 4095 do;
    end; { of wait }
    Procedure check_vectors(phase : integer);
    var i,j,k : integer;
    begin
        writeln('checking test vectors phase ',phase);
        stop_processor;
        address := 2;
        case phase of
            0:
                for k := 0 to last_op do
                    for j := 0 to last_s do
                        for i := 0 to last_r do
                            begin
                                check_fpu_vector(address,r[i],s[j],op[k]);
                                address := address + 1;
                            end;
                        end;
                    end;
                1:
                    for k := 0 to last_op do
                        for i := 0 to last_r do
                            for j := 0 to last_s do
                                begin
                                    check_fpu_vector(address,r[i],s[j],op[k]);
                                    address := address + 1;
                                end;
                            end;
                        end;
                    2:
                        for j := 0 to last_s do
                            for i := 0 to last_r do
                                for k := 0 to last_op do
                                    begin
                                        check_fpu_vector(address,r[i],s[j],op[k]);
                                        address := address + 1;
                                    end;
                                end;
                            end;
                        end;
        end; { of check_vectors }
    procedure check_results(phase : integer);
    var i,j,k : integer;
    begin
        writeln('checking test result');
        start_processor;
        wait;
        address := 2;
        stop_processor;
        case phase of
            0 :
                for k := 0 to last_op do
                    for j := 0 to last_s do
                        for i := 0 to last_r do
                            begin
                                if not odd(k) then
                                    check_fpu_result(address,r[i],s[j],op[k]);
                                    address := address + 1;
                                end;
                            end;
                        end;
                    end;
                1 :
                    for k := 0 to last_op do
                        for i := 0 to last_r do
                            for j := 0 to last_s do
                                begin
                                    if not odd(k) then
                                        check_fpu_result(address,r[i],s[j],op[k]);
                                        address := address + 1;
                                    end;
                                end;
                            end;
                        end;
                2 :
                    for j := 0 to last_s do

```

```

        for i := 0 to last_r do
            for k := 0 to last_op do
                begin
                    if not odd(k) then
                        check_fpu_result(address, r[i], s[j], op[k]);
                        address := address + 1;
                    end;
                end;
            end; { of check_result }
        procedure test_random;
        var i, j : integer;
        begin
            writeln('-- random pattern test');
            stop_processor;
            writeln('generate vectors');
            j := 0;
            for i := 2 to 4090 do
                begin
                    r[i] := random($ffff);
                    r[i] := (r[i] shl 16) or random($ffff);
                    s[i] := r[i];
                    write_fpu_vector(i, r[i], s[i], op[j]);
                    if j = last_op then j := 0 else j := j+1;
                end;
            end;
            writeln('checking vectors');
            j := 0;
            for i := 2 to 4090 do
                begin
                    check_fpu_vector(i, r[i], s[i], op[j]);
                    if j = last_op then j := 0 else j := j+1;
                end;
            end;
            start_processor;
            wait;
            writeln('checking fpu results');
            stop_processor;
            j := 0;
            for i := 2 to 4090 do
                begin
                    if not odd(j) then
                        check_fpu_result(i, r[i], s[i], op[j]);
                    if j = last_op then j := 0 else j := j+1;
                    end;
                end;
            end; { of test_random }
        procedure test_logical;
        var
            phase : integer;
        begin
            writeln('-- xor/and/or/passR tests --');
            test_case := logical;
            op[0] := fpu_passR; op[1] := not op[0];
            op[2] := fpu_and;   op[3] := not op[2];
            op[4] := fpu_or;    op[5] := not op[4];
            op[6] := fpu_xor;   op[7] := not op[6];
            op[8] := fpu_xor;   op[9] := not op[6];
            last_op := 9;
            test_cycle := 0;
            repeat
                writeln('---- logical test cycle ', test_cycle, ' ---');
                test_cycle := test_cycle + 1;
                writeln('-- fixed pattern test');
                r[0] := $00000000; r[1] := $ffffff; r[2] := $55555555; r[3] := $aaaaaaaa;
                r[4] := $ffffff; r[5] := $00000000; r[6] := $12345678; r[7] := $9abcdef0;
                last_r := 7;
                s[0] := $00000000; s[1] := $ffffff; s[2] := $55555555; s[3] := $aaaaaaaa;
                s[4] := $ffffff; s[5] := $00000000; s[6] := $12345678; s[7] := $9abcdef0;
                last_s := 7;
                for phase := 0 to 2 do
                    begin
                        generate_vectors(phase);
                        check_vectors(phase);
                        check_results(phase);
                    end;
                end;
                test_random;
            until (continuous <> 'y') and (continuous <> 'Y');
        end; { of test_logical }
        procedure test_iadd;
        var
            phase : integer;
        begin

```

```

test_case := iadd;
test_cycle := 1;
writeln('begin integer add/sub/rsub tests');

op[0] := fpu_add; op[1] := not op[0];
op[2] := fpu_sub; op[3] := not op[2];
op[4] := fpu_rsub; op[5] := not op[4];
last_op := 5;

test_cycle := 1;
repeat
  writeln('---- integer add test cycle ',test_cycle,' ---');
  test_cycle := test_cycle + 1;
  writeln('-- fixed pattern test');
  r[0] := $00000000; r[1] := $00ffffff; r[2] := $00555555; r[3] := $00aaaaaa;
  r[4] := $00000001; r[5] := $00123456; r[6] := $00abcdef; r[7] := $80ffffff;
  r[8] := $80555555; r[9] := $80aaaaaa; r[10] := $80000001; r[11] := $80123456;
  r[12] := $80abcdef;
  last_r := 12;
  s[0] := $00000000; s[1] := $00ffffff; s[2] := $00555555; s[3] := $00aaaaaa;
  s[4] := $00000001; s[5] := $00123456; s[6] := $00abcdef; s[7] := $80ffffff;
  s[8] := $80555555; s[9] := $80aaaaaa; s[10] := $80000001; s[11] := $80123456;
  s[12] := $80abcdef;
  last_s := 12;
  for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
  test_random;
until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_fix }

procedure test_imult;
var
  phase : integer;
begin
  test_case := imult;
  test_cycle := 1;
  writeln('begin integer mult tests');

  op[0] := fpu_mult; op[1] := not op[0];
  last_op := 1;

  test_cycle := 1;
  repeat
    writeln('---- integer mult test cycle ',test_cycle,' ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $00000000;
    r[1] := $00ffffff; r[2] := $00aaaaaa; r[3] := $00555555;
    r[4] := $00000fff; r[5] := $00000aaa; r[6] := $00000555; r[7] := $00ffffff;
    r[8] := $80000000;
    r[9] := $80ffffff; r[10] := $80aaaaaa; r[11] := $80555555;
    r[12] := $80000fff; r[13] := $80000aaa; r[14] := $80000555; r[15] := $80ffffff;
    last_r := 15;
    s[0] := $00000000;
    s[1] := $00ffffff; s[2] := $00aaaaaa; s[3] := $00555555;
    s[4] := $00000fff; s[5] := $00000aaa; s[6] := $00000555; s[7] := $00ffffff;
    s[8] := $80000000;
    s[9] := $80ffffff; s[10] := $80aaaaaa; s[11] := $80555555;
    s[12] := $80000fff; s[13] := $80000aaa; s[14] := $80000555; s[15] := $80ffffff;
    last_s := 15;
    for phase := 0 to 2 do
      begin
        generate_vectors(phase);
        check_vectors(phase);
        check_results(phase);
      end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_imult }

procedure test_fmult;
var
  phase : integer;
begin
  test_case := fmult;
  test_cycle := 1;
  writeln('begin float mult tests');

```

```

op[0] := fpu_fmult; op[1] := not op[0];
last_op := 1;

test_cycle := 1;
repeat
  writeln('---- floating mult test cycle ',test_cycle,' ---');
  test_cycle := test_cycle + 1;
  writeln('-- fixed pattern test');
  r[0] := $00000000;
  r[1] := $3fffffff; r[2] := $3faaaaaa; r[3] := $3f555555;
  r[4] := $00000fff; r[5] := $00000aaa; r[6] := $7f000555; r[7] := $7fffffff;
  r[8] := $08000000;
  r[9] := $bfffffff; r[10] := $bfaaaaaa; r[11] := $bf555555;
  r[12] := $80000fff; r[13] := $80000aaa; r[14] := $ff000555; r[15] := $ffffffff;
  last_r := 15;
  s[0] := $00000000;
  s[1] := $3fffffff; s[2] := $3faaaaaa; s[3] := $3f555555;
  s[4] := $00000fff; s[5] := $00000aaa; s[6] := $7f000555; s[7] := $7fffffff;
  s[8] := $80000000;
  s[9] := $bfffffff; s[10] := $bfaaaaaa; s[11] := $bf555555;
  s[12] := $80000fff; s[13] := $80000aaa; s[14] := $ff000555; s[15] := $ffffffff;
  last_s := 15;
  for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
  test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_fmult }

procedure test_fadd;
var
  phase : integer;
begin
  test_case := fadd;
  test_cycle := 1;
  writeln('begin floating point add tests');

  op[0] := fpu_fadd; op[1] := not op[0];
  op[2] := fpu_fsub; op[3] := not op[2];
  op[4] := fpu_frsb; op[5] := not op[4];
  last_op := 5;

  test_cycle := 1;
  repeat
    writeln('---- floating add test cycle ',test_cycle,' ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $3f000000;
    r[1] := $3fffffff; r[2] := $3faaaaaa; r[3] := $3f555555; r[4] := $3f000001;
    r[5] := $7f000000; r[6] := $2a800000; r[7] := $55000000; r[8] := $00800000;
    r[9] := $bf000000;
    r[10] := $bfffffff; r[11] := $bfaaaaaa; r[12] := $bf555555; r[13] := $bf000001;
    r[14] := $ff000000; r[15] := $aa800000; r[16] := $d5000000; r[17] := $80800000;
    last_r := 17;
    s[0] := $3f000000;
    s[1] := $3fffffff; s[2] := $3faaaaaa; s[3] := $3f555555; s[4] := $3f000001;
    s[5] := $7f000000; s[6] := $2a800000; s[7] := $55000000; s[8] := $00800000;
    s[9] := $bf000000;
    s[10] := $bfffffff; s[11] := $bfaaaaaa; s[12] := $bf555555; s[13] := $bf000001;
    s[14] := $ff000000; s[15] := $aa800000; s[16] := $d5000000; s[17] := $80800000;
    last_s := 17;
    for phase := 0 to 2 do
      begin
        generate_vectors(phase);
        check_vectors(phase);
        check_results(phase);
      end;
    test_random;
    until (continuous <> 'y') and (continuous <> 'Y');
  end; { of test_fadd }

procedure test_shift;
var
  phase : integer;
  var i : integer;
begin
  test_case := shift;
  test_cycle := 1;
  writeln('begin ror/rol/shr/shl tests');

```

```

op[0] := fpu_ror; op[1] := not op[0];
op[2] := fpu_rol; op[3] := not op[2];
op[4] := fpu_shr; op[5] := not op[4];
op[6] := fpu_shl; op[7] := not op[6];
last_op := 7;

test_cycle := 1;
repeat
  writeln('---- ror/rol/shr/shl test cycle ', test_cycle, ' ---');
  test_cycle := test_cycle + 1;
  writeln('-- fixed pattern test');
  r[0] := $00000000;
  r[1] := $ffffff; r[2] := $aaaaaaaa; r[3] := $55555555; r[4] := $01234567;
  r[5] := $89abcdef;
  last_r := 5;
  for i := 0 to 31 do s[i] := i;
  last_s := 31;
  for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
  test_random;
until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_shift }

procedure test_special0;
var
  phase : integer;
begin
  test_case := special;
  test_cycle := 1;
  writeln('Testing Pack Exp and Float');
  op[0] := fpu_pack; op[1] := not op[0];
  op[2] := fpu_float; op[3] := not op[0];
  last_op := 3;
  r[0] := 1; r[1] := 4; r[2] := 16; r[3] := 64; r[4] := 256;
  r[5] := $ffffff;
  r[6] := $80000001; r[7] := $80000004; r[8] := $80000010;
  r[9] := $80000040; r[10] := $80000100; r[11] := $80ffffff;
  last_r := 11;
  s[0] := 0; s[1] := not s[0];
  last_s := 1;
  repeat
    writeln('---- special test 0 cycle ', test_cycle, ' ---');
    test_cycle := test_cycle + 1;
    for phase := 0 to 2 do
      begin
        generate_vectors(phase);
        check_vectors(phase);
        check_results(phase);
      end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_special0 }

procedure test_special1;
var
  phase : integer;
begin
  test_case := special;
  test_cycle := 1;
  writeln('Testing seed, unexp and unman, rootexp, rootman');
  op[0] := fpu_seed; op[1] := not op[0];
  op[2] := fpu_unexp; op[3] := not op[0];
  op[4] := fpu_unman; op[5] := not op[0];
  op[6] := fpu_rootexp; op[7] := not op[0];
  op[8] := fpu_rootman; op[9] := not op[0];
  last_op := 9;
  r[0] := $00000000;
  r[1] := $3ffffff;
  r[2] := $00000fff;
  r[3] := $7f000555;
  r[4] := $08000000;
  r[5] := $bffffff;
  r[6] := $80000fff;
  r[7] := $ff000555;
  last_r := 7;
  s[0] := $00000000;
  s[1] := $3ffffff;
  s[2] := $00000fff;

```

```

s[3] := $7f000555;
s[4] := $08000000;
s[5] := $bfffffff;
s[6] := $80000fff;
s[7] := $ff000555;
last_s := 7;
repeat
  writeln('---- special test 1 cycle ',test_cycle,' ---');
  test_cycle := test_cycle + 1;
  for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
  test_random;
until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_special1 }

procedure test_special2;
var
  phase : integer;
begin
  test_case := special;
  test_cycle := 1;
  writeln('Testing Round and Trunc/Int');
  op[0] := fpu_round; op[1] := not op[0];
  op[2] := fpu_int_r; op[3] := not op[0];
  op[4] := fpu_int_s; op[5] := not op[0];
  last_op := 5;
  r[0] := $00000000;
  r[1] := $3fffffff;
  r[2] := $46000fff;
  r[3] := $3f000555;
  r[4] := $4b000000;
  r[5] := $bfffffff;
  r[6] := $c6000fff;
  r[7] := $bf000555;
  last_r := 7;
  s[0] := $00000000;
  s[1] := $3fffffff;
  s[2] := $46000fff;
  s[3] := $3f000555;
  s[4] := $4b000000;
  s[5] := $bfffffff;
  s[6] := $c6000fff;
  s[7] := $bf000555;
  last_s := 7;
repeat
  writeln('---- special test 2 cycle ',test_cycle,' ---');
  test_cycle := test_cycle + 1;
  for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
  test_random;
until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_special2 }

procedure test_special3;
var
  phase : integer;
begin
  test_case := special;
  test_cycle := 1;
  writeln('Testing Special Sign Manipulation');
  op[0] := fpu_sin_sgn; op[1] := not op[0];
  op[2] := fpu_odd_neg; op[3] := not op[0];
  op[4] := fpu_chq_sgn; op[5] := not op[0];
  op[6] := fpu_tan_sgn; op[7] := not op[0];
  last_op := 7;
  r[0] := $00000000;
  r[1] := $00000001;
  r[2] := $80000010;
  r[3] := $80000001;
  r[4] := $3f800000;
  r[5] := $3f800001;
  r[6] := $bf800010;
  r[7] := $bf800001;
  last_r := 7;

```



```

s[0] := $00000000;
s[1] := $00000001;
s[2] := $80000010;
s[3] := $80000001;
s[4] := $3f800000;
s[5] := $3f800001;
s[6] := $bf800010;
s[7] := $bf800001;
last_s := 7;
repeat
  writeln('---- special test 3 cycle ',test_cycle,' ---');
  test_cycle := test_cycle + 1;
  for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
  test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_special3 }

procedure substitute_word;
var start_address, data : word;
    xd : string;
begin
  write('start address ? '); readln(start_address);
  if odd(start_address) then start_address := start_address - 1;
  repeat
    write(word_to_hex(start_address),':',word_to_hex(mread(bank,start_address)));
    write(' -> '); readln(xd);
    if xd <> 'quit' then
      begin
        data := hex_to_word(xd);
        mwrite(bank,start_address,data);
        verify(bank,address,mread(bank,start_address),data);
        start_address := start_address + 2;
      end;
    until xd = 'quit';
end; { of substitute_word }

procedure display_word;
var i,start_address, no_word : word;
begin
  write('start address ? '); readln(start_address);
  if odd(start_address) then start_address := start_address - 1;
  write('no of words ? '); readln(no_word);
  writeln('memory bank ',bank);
  i := start_address;
  while i <= (start_address + no_word*2) do
    begin
      writeln(word_to_hex(i),':',word_to_hex(mread(bank,i)));
      i := i + 2;
    end;
end; { of display_word }

procedure select_bank;
begin
  write('bank : ',bank,' -> '); readln(bank);
end; { of select_bank }

procedure t;
begin
  stop_processor;
  for i := 0 to 32 do
    begin
      mwrite(4,i shl 1,i);
      verify(4,i shl 1,mread(4,i shl 1),i);
    end;
  start_processor;
end;

procedure toggle_debug;
begin
  if debug = true then
    begin
      debug := false;
      writeln('debugging off');
    end
  else
    begin
      debug := true;
    end
end;

```

```

        writeln('debugging on');
    end;
end;

Procedure test_all;
var local_cont: char;
begin
    local_cont := continuous;
    continuous := 'n';
    count := 1;
    repeat
        writeln('*** Test all functions *** Cycle ',count,' (errors = ',no_error,') ***');
        test_logical;
        writeln('*****');
        test_iadd;
        writeln('*****');
        test_fadd;
        writeln('*****');
        test_imult;
        writeln('*****');
        test_fmult;
        writeln('*****');
        test_shift;
        writeln('*****');
        test_special0;
        writeln('*****');
        test_special1;
        writeln('*****');
        test_special2;
        writeln('*****');
        test_special3;
        writeln('*****');
        count := count + 1;
    until (local_cont <> 'y') and (local_cont <> 'Y');
    continuous := local_cont;
end;

begin
    debug := false;
    continuous := 'n';
    stop_on_error := 'y';
    no_error := 0;
    writeln('FPU Test Monitor');
    stop_processor;
    while true do
        begin
            write('>'); readln(command);
            if command = 'help' then
                begin
                    writeln('tmem    : test memory');
                    writeln('tlog    : test xor/and/or/passR');
                    writeln('tiadd    : test integer add/sub/rsub');
                    writeln('timult   : test integer mult');
                    writeln('tfadd    : test floating point add');
                    writeln('tfmult   : test floating point mult');
                    writeln('tshift   : test ROR/ROL/SHR/SHL');
                    writeln('tspec0   : test pack exp & float ');
                    writeln('tspec1   : test seed, unp_exp, unp_man, root_exp, & root_man');
                    writeln('tspec2   : test round & trunc');
                    writeln('tspec3   : test sign manipulation');
                    writeln('tall     : test all of the above');
                    writeln('dsoe     : do not stop on error');
                    writeln('soe      : stop on error');
                    writeln('start    : start testing');
                    writeln('stop     : stop testing');
                    writeln('sb       : select memory bank');
                    writeln('dw       : display memory word');
                    writeln('sw       : substitute memory word');
                    writeln('cont     : set testing mode to continuous');
                    writeln('single   : set testing mode to single');
                    writeln('debug    : toggle debug setting');
                    writeln('quit     : quit FPU Test Monitor');
                end
            else
                if command = 'tmem' then test_memory
                else
                    if command = 'tlog' then test_logical
                    else
                        if command = 'tiadd' then test_iadd
                        else
                            if command = 'tfadd' then test_fadd
                            else
                                ;
                            end
                        end
                    end
                end
            end
        end
    end
end

```

```

if command = 'timult' then test_imult
else
if command = 'tfmult' then test_fmult
else
if command = 'tshift' then test_shift
else
if command = 'tspec0' then test_special0
else
if command = 'tspec1' then test_special1
else
if command = 'tspec2' then test_special2
else
if command = 'tspec3' then test_special3
else
if command = 'tall' then test_all
else
if command = 'dsoe' then stop_on_error := 'n'
else
if command = 'soe' then stop_on_error := 'y'
else
if command = 'start' then start_processor
else
if command = 'stop' then stop_processor
else
if command = 'sw' then substitute_word
else
if command = 'dw' then display_word
else
if command = 'sb' then select_bank
else
if command = 't' then t
else
if command = 'debug' then toggle_debug
else
if command = 'quit' then exit
else
if command = 'cont' then
begin
continuous := 'y';
writeln('Next test specified will be repeated indefinitely');
end
else
if command = 'single' then
begin
continuous := 'n';
writeln('Next test specified will not be repeated');
end
else
if command = 'exit' then exit
end;
end.

```